JetBox Linux SDK

User Manual

www.korenix.com

0.0.3

Copyright Notice

Acknowledgments

Table of Content

Chapter 1 Overview

JetBox Linux SDK (software development kit) has all the required software and utilities for you to develop your own applications. There are two folders in JetBox Linux SDK. One folder is for application source code and the other one is for Linux tool chain.

Toolchain

The Toolchain means a suite of cross compilers, other tools, and the libraries and header files that are necessary to compile your application. These tool chain components must be installed on your computer running Linux.

Application source code (ap_src)

This folder is for your application source code storage. One sample code inside is for your reference.

Chapter 2 Applied Korenix Models

There are several different versions of compilers for different models of the JetBox series. Kindly refer to the mapping table to choose the correct compiler for your application.

Product Name	GCC Compiler Version	SDK
JetBox 8100	4.1.1	jetbox8100_sdk-k0.4.tgz
JetBox 8210	4.2.1	jetbox8210-sdk-k0.4.tgz
JetOS93 JetBox 9300 series JetOS93Lite JetBox 5300 series JetBox 3300 series	3.4.6	jetbox9300_sdk-k0.6.tgz
JetOS95 JetBox 9500 series JetBox 9400 series JetBox 5400 series	4.2.4	Jetbox9500_sdk-k0.5.tgz

⚠ **Notice 1:** "k0.4"of the SDK file name is the version of the SDK. Please contact Korenix for latest version.

Chapter 3 Installing SDK

JetBox SDK must be installed on your host computer running Linux. We have confirmed that **Fedore Core 9** Linux distribution can be used to install the toolchain. Using the command **tar** to uncompress the SDK archive at the directory where you want to complete the installation and all the files inside the SDK will be put at its corresponding folders.

Take the JetBox 9300 SDK as an example:

cd ~
tar zfx jetbox9300_sdk-k0.6.tgz
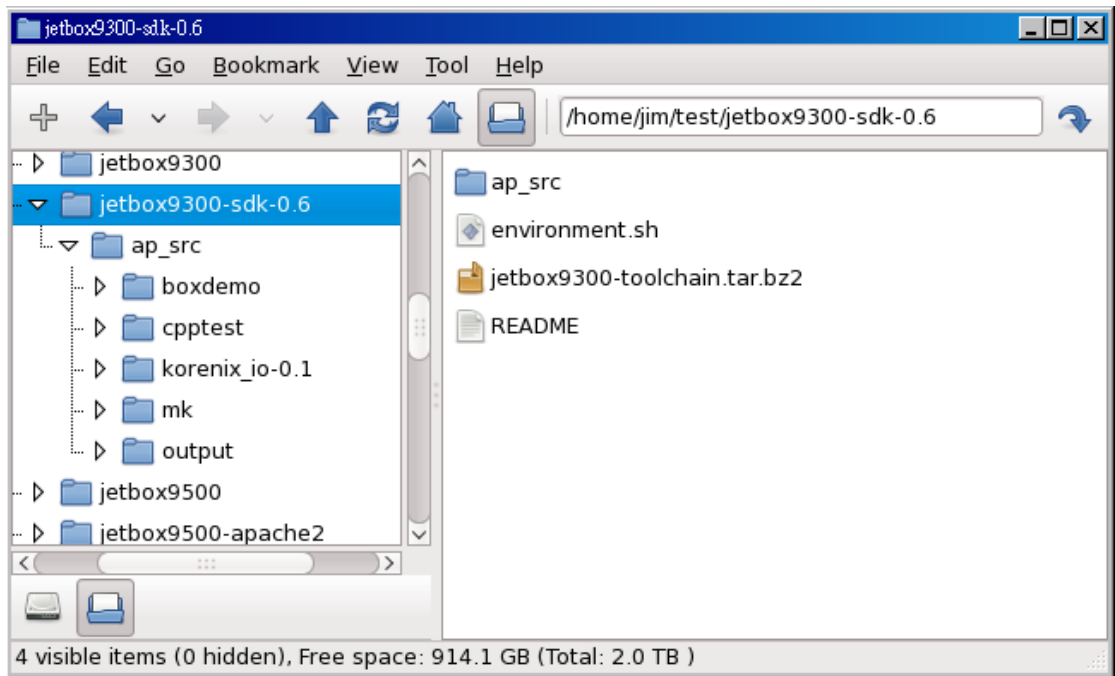
Snapshot 1: The directory of the JetBox 9310 SDK

The ap_src directory is the sample code directory. The README file has the detail instructions guide you to extract Toolchain.tar.bz2 to the right directory.

Chapter 4 SDK Content

4-1   ToolChain

The toolchain consists of the GNU binutils, compiler set (GCC) and debugger (Insight for Linux). The toolchain includes the C and C++ compilers.

Utility	Use
addr2line	Converts addresses into line numbers within original source files
ar	GNU assembler
as	Creates and manipulates archive content

| Utility	Use
c++ filt	Converts low-level, mangled assembly labels resulting from overloaded C++ functions to their user-level names
ld	GNU linker
objcopy	Copies and translates object files
objdump	Displays information about the content of object files
ranlib	Generates an index to the content of an archive
readelf	Displays information about an ELF format object file
size	Lists the sizes of sections within an object file
strings	Prints the strings of printable characters in object files
strip	Strips symbols from object files

Chart 1: Binutils Utilities in the JetBox SDK

Following is the tool name list of GCC (GNU project C compiler) and C++ (GNU project C++ compiler).

If the JetBox series are arm based Linux computer, then $(TARGET) = arm-linux-, therefore the cross-compiler name will be arm-linux-gcc or arm-linux-c++.

CROSS_COMPILE	=$(TARGET)-
AS	=$(CROSS_COMPILE)as
AR	=$(CROSS_COMPILE)ar
CC	=$(CROSS_COMPILE)gcc
CPP	=$(CC) -E
LD	=$(CROSS_COMPILE)ld
NM	=$(CROSS_COMPILE)nm
OBJCOPY	=$(CROSS_COMPILE)objcopy
OBJDUMP	=$(CROSS_COMPILE)objdump
RANLIB	=$(CROSS_COMPILE)ranlib
READELF	=$(CROSS_COMPILE)readelf
SIZE	=$(CROSS_COMPILE)size
STRINGS	=$(CROSS_COMPILE)strings
STRIP	=$(CROSS_COMPILE)strip

Chart 2: Tool names of C & C++ compilers
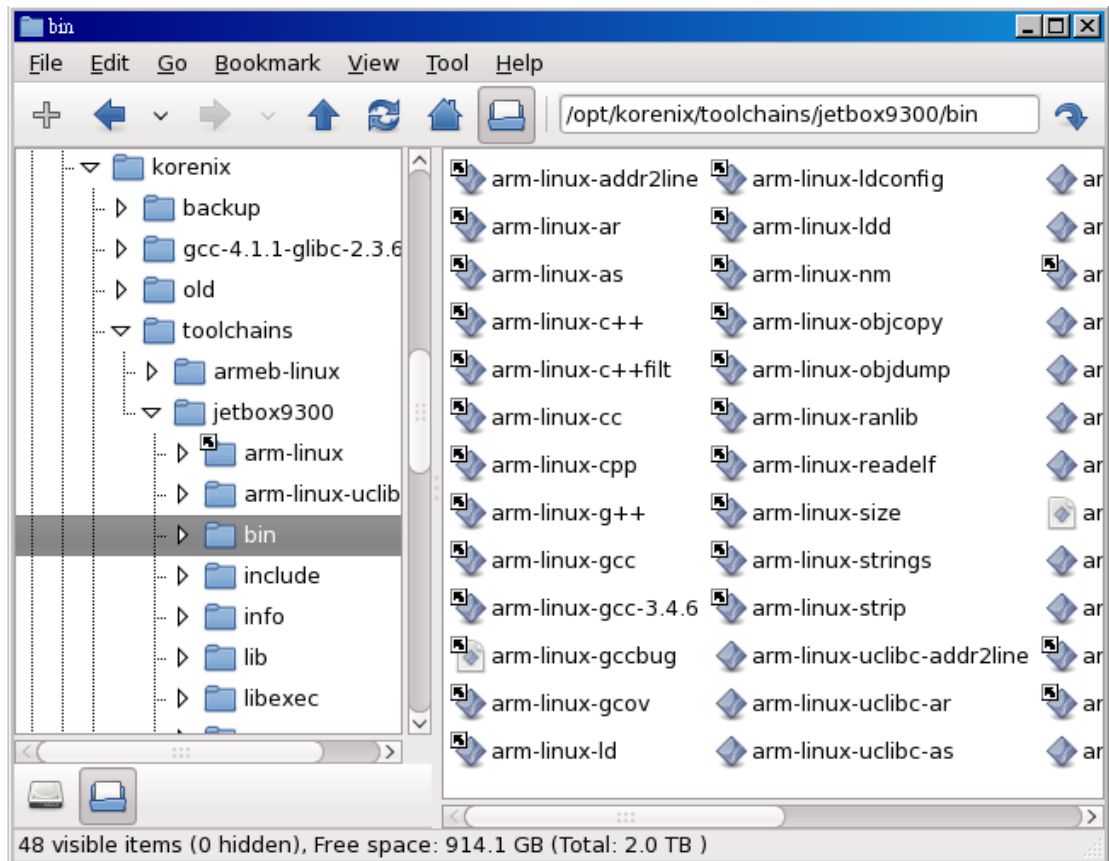
Snapshot 2: Cross compiler tool list

4-2   Application Source Code

There are several example applications at the directory **ap_src** for your programming reference.

Application Name	Description	Files or direction
diocfg	IO control application code	korenix_io-0.1/
boxdemo	IO sequence control for JetBox DemoBox	boxdemo-0.1/
sum	C++ code for calculate total of two integer.	sum.cpp

Chart 3: Demo applications at ap_src directory

"diocfg" is used to get or set the status of digital inputs or outputs. Please refer to following snapshot for the usage description.
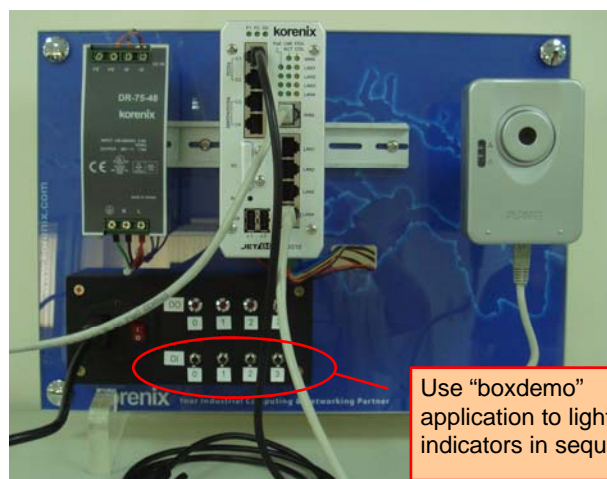
Snapshot 3: diocfg usage

"boxdemo" is used in the JetBox demo box to light the indicators connected to the digital output 0, 1, 2, and 3 in sequence (the time slot between two indicators lighting is 1 sec). Just show an easy programming example to control the digital output.

Snapshot 4: Boxdemo application for the JetBox demo box

"sum" is used to test the C++ cross compiler. It simply shows the sum of two integers. Following is the snapshot of "sum."

Snapshot 5: sum usage

Chapter 5 Writing Your Own Applications

Writing applications for the JetBox is the same as writing applications for a Linux PC. The better way to develop your own application is to write and test your own applications on a Linux PC first and then port the application from the Linux PC to the JetBox.

5-1   Steps to Develop a Application

Following is the steps to develop an application on a Linux PC and port to the JetBox.

Developing on a Linux PC

1. Create a directory under **ap_src/** and put the source code into this directory.
2. Create a makefile (a controlling file) in **ap_src/** for this application. The makefile extension must be "**.mk**".
3. Use the command **make** under the directory **ap_src/**. The application will be built and the executable binary will be generated in the directory **ap_src/**.

Porting to the JetBox

4. Put the executable binary into the JetBox SD card or CF card depending of the JetBox models.

In the JetBox SDK, there is an example code for your reference

Korenix_io-01/ is the sample code of accessing the GPIO of the JetBox 9300/9310

5-2   Steps to Compile Source Codes

Following is an example of the JetBox 9310 at the cross-compiler direct to show you how to compile source codes.

```
/opt/korenix/toolchains/jetbox9300/bin/arm-linux-gcc test.c –o test
```

The file "test.c" is the application source code and the file "test" is the executable binary.

This test application was used in the JetBox 9310 demonstration. The scenario is to control both the PoE LAN ports and the digital outputs through a specific COM port. And the purpose of this demo application is to show the programming and management ability of the JetBox 9310.

In this demo, a card reader is connected to COM1. A PoE IP cam is connected to LAN1. 3 indicators are connected to the digital output of the JetBox 9310. The application is stored in a SD card and the SD card is inserted into the JetBox 9310. After executing the test application, using different magnetic cards can control the PoE IP cam and the indicators.
Card1: the serial number is recognized by test application to enable the PoE IP cam and indicator 1.
Card2: the serial number is recognized by test application to disable the PoE IP cam and indicator 2.
Card3: This is an unknown serial number. No actions of PoE IP cam and enable the indicator 3.
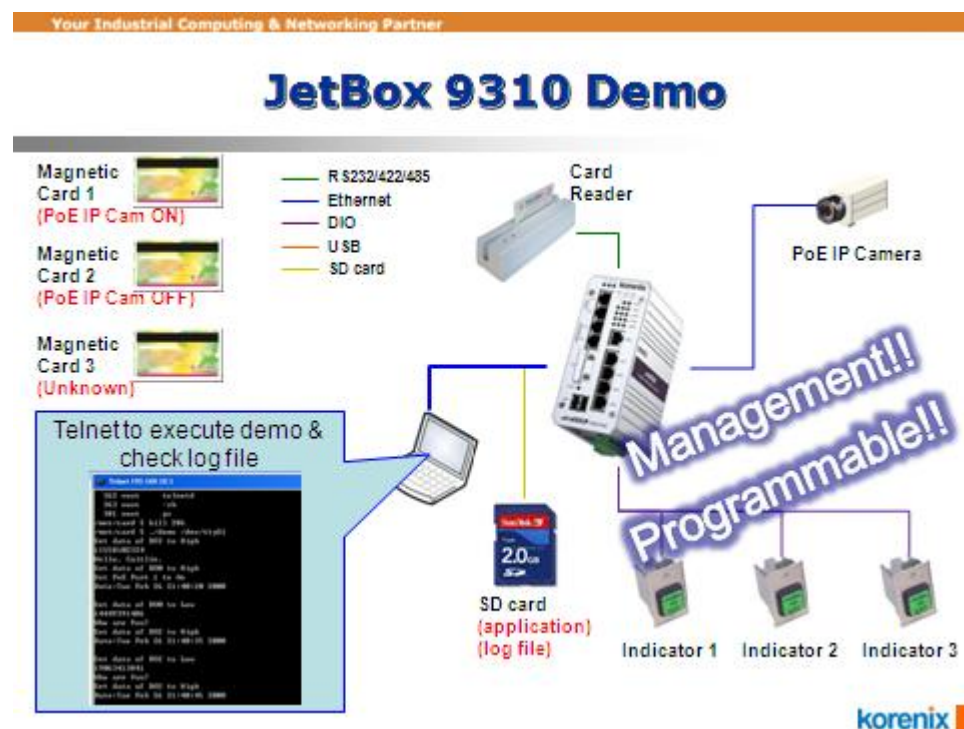

Chart 4: The illustration of the test application of the JetBox 9310.

⚠️**Notice 2:** You can use the auto-run function of the JetBox 9300/9310 to execute an application automatically. Please refer to the manual of the JetBox 9300/9310 autorun.

5-3   Deploying Your Application

After you built your own application, you can use several ways to deploy your application to the JetBox depending on different models.

5-3-1   The JetBox 9300/9310

Copy the executable binaries from your host Linux PC into a SD card. Inset the card into the JetBox. Use the autorun function to execute applications.

⚠️**Notice 3:** Refer to the autorun user manual to run your own applications through autorun function.

5-3-2   The JetBox 8100/8210

Copy the executable binaries from your host Linux PC into the system CF card. Inset the card into the JetBox for execution.
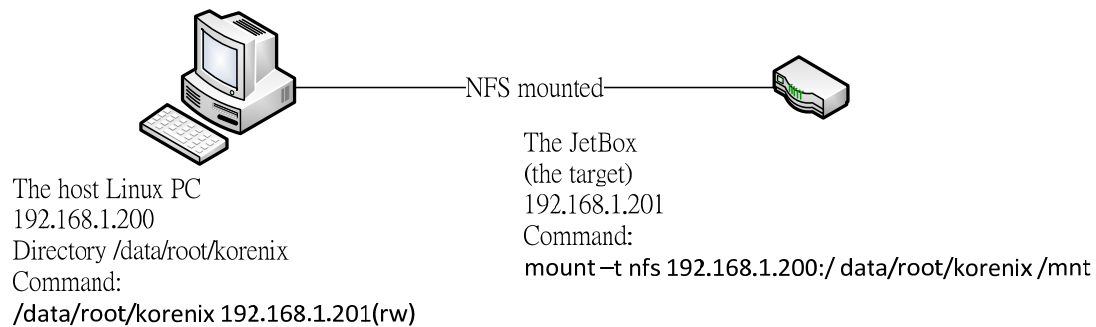
You could copy the executable binaries into a USB flash disk, put the USB flash disk into the JetBox, mount the USB flash disk, and copy the executable binaries from the USB flash disk to the JetBox.

Or you could use the remote components of the JetBox to facilitate development, such as TFTP. The directory with executable binaries could be NFS mounted instead of being on the storage media in the JetBox. Using an NFS mounted directory is perfect during development, because it avoids having to constantly copy program modifications between the host PC and the JetBox (the target).

Following is an example of NSF mounted.

The Host PC IP address: 192.168.1.200

The JetBox (the target) IP address: 192.168.1.201

The host Linux PC
192.168.1.200
Directory /data/root/korenix
Command:
/data/root/korenix 192.168.1.201(rw)

The JetBox
(the target)
192.168.1.201
Command:
mount –t nfs 192.168.1.200:/ data/root/korenix /mnt

The NFS server setting on your host PC

You need to have the NFS utilities installed on your host PC. An nfs-utils package is part of your distribution. With the nfs-utils package installed, you need to make sure that the appropriate configuration files are present and the corresponding services are started. The main file we need to configure for the NFS server is **/etc/exports**. Entries in this file describe the directories that each host or set of hosts can access. Here's an example:

/data/root/korenix 192.168.1.201(rw)

This entry states that the target machine with address 192.168.1.201 has read and writes (rw) access to the **/data/root/korenix** directory

The NFS mounted on the JetBox (the target)

In the target platform, with kernel support for NFS enabled, you can mount a NFS filesystem on /mnt; for example:

mount –t nfs 192.168.1.200:/ data/root/korenix /mnt

This entry states that the NFS server with address 192.168.1.200 has exported to the **/data/root/korenix** directory

Chapter 6 Appendix

6-1   Chart Index

6-2   Customer Service

Korenix Technologies Co., Ltd.

Business service: sales@korenix.com

Customer service: koreCARE@korenix.com